Vers un partitionnement heuristique pour le mapping de data - path sur FPGA reconfiguré dynamiquement

Camel TANOUGAST, Yves BERVILLER, Serge WEBER

Laboratoire d' Instrumentation Electronique de Nancy - L. I. E. N.

Faculté des Sciences de Nancy – BP 239

54506 Vandoeuvre - lès – Nancy.

{tanougast, yves.berviller, serge.weber}@lien.u-nancy.fr

Résumé - Le problème traite du partitionnement temporel des algorithmes de TSI ayant une contrainte temps réel, en vue de leurs implantations optimisées sur architecture reconfigurable dynamiquement. Nous présentons une méthode, pouvant être appliquée heuristiquement, pour déterminer les étapes de la reconfiguration dynamique (RD) et un partitionnement adapté à l'implantation d'un data–path (chemin de données). L'évaluation de certaines caractéristiques liées à la technologie et la proposition d'une méthode appliquée heuristiquement, nous permettent un partitionnement optimisant les ressources matérielles en RD.

Abstract *– This paper discuss about the temporal partitioning problem of TSI algorithms time constrained, for implementation optimized in run-time reconfiguration on FPGA. We present a methodology which can be applied in heuristic way for determined the reconfiguration steps and a partitioning adapted for implementation of algorithms data – path. The evaluation of some parameters of technology and the proposition of a method allows to obtain a partitioning optimizing the hardware logic resources in dynamic reconfiguration.*

1. Introduction

La reconfiguration dynamique (RD) des FPGAs consiste à exécuter successivement une séquence d'algorithmes sur le même circuit. Le but est de permuter différents algorithmes sur la même structure matérielle en reconfigurant sous une contrainte de temps le FPGA avec un partitionnement et un ordonnancement définis.

Les systèmes embarqués peuvent tirer plusieurs avantages de l'emploi de FPGAs. Le plus évident est la possibilité de réadapter fréquemment les fonctions logiques matérielles. Mais nous pouvons également utiliser la possibilité d'allocation dynamique des ressources dans le but d'instancier uniquement les opérateurs nécessaires à un instant donné. Cela permet d'améliorer le rendement du silicium en réduisant la surface de la matrice reconfigurable du FPGA [1]. Cependant le mécanisme d'une décomposition optimale des algorithmes (partitionnement) est un aspect qui est relativement peu traité. En général, les stratégies d'implantations en RD d'algorithmes TSI sont réalisées en implantant successivement dans le FPGA les différents opérateurs haut niveau les constituant [2, 3]. Or, le rendement obtenu n'est pas toujours optimal par rapport aux ressources spatio-temporelles disponibles. D'autres permettent de réduire les durées de reconfiguration et d'exécution de l'application mais ne prennent pas en compte la notion de contrainte de temps et ne guident pas le partitionnement [4]. Contrairement à ces approches qui consistent à obtenir un temps d'exécution court qui respecte une contrainte de surface, nous souhaitons optimiser la surface logique en exploitant entièrement la contrainte de temps. Dans ce but, nous cherchons une partition indépendante des opérateurs haut niveau de l'application en considérant l'algorithme à son plus bas niveau et dans sa globalité.

2. Formulation du problème.

Un data-path peut être considéré comme un ensemble donné d'opérateurs $\{O_1,\ldots,O_n\} \in$ O d'un graphe flot de données (GFD) $G(O,C)$ caractérisés par un temps d'exécution t_{Oi} et une surface S_{Vi} qui respecte les contraintes de dépendance de données $\{C_1,\ldots,C_n\} \in$ C entre les opérateurs. La tâche du partitionnement pour le mapping RD en E étapes peut être spécifiée comme la recherche des sous-ensembles d'opérateurs $\{O_j,\ldots,O_k\}$ physiquement présents dans le FPGA et qui minimise la surface totale S_T du FPGA tout en satisfaisant la contrainte de temps T. Nous considérons uniquement les opérateurs arithmétiques et logiques élémentaires (additionneurs, soustracteurs, multiplexeurs, registres etc.) de l'algorithme. Nous analysons le GFD dans le but de déduire les valeurs de certains paramètres, en particulier les ressources nécessaires (en terme de surface) et le temps d'exécution maximal pour chaque opérateur. Nous déduisons une estimation du nombre d'étapes. Cette première évaluation est donnée par l'équation suivante :

$$E = \frac{T}{N \cdot K \cdot to_{max} + \frac{S_{G(O,C)}}{V_C}} \qquad (1)$$

Où **E** est le nombre de reconfiguration minimal (E ∈ IN), **N** est le nombre de données à traiter, **T** est la limite du temps de traitement (contrainte de temps), to_{max} est le temps d'exécution maximal parmi les opérateurs du graphe flot de données (sans routage), K est un coefficient qui prend en compte le délai de routage entre les opérateurs, V_C est la vitesse de reconfiguration caractérisant la famille du FPGA utilisé et $S_{G(O,C)}$ est le nombre de cellules logiques nécessaires pour implanter globalement le data-path. Cette évaluation est donc obtenue avec le temps de configuration maximal et le temps d'exécution de l'opérateur le plus lent de l'algorithme. Dans une étape, le temps élémentaire de

traitement est fixé par le temps maximal entre deux opérateurs successifs du graphe $G(O,C)$. Car nous synchronisons les données avec des registres. C'est pourquoi, le temps de traitement élémentaire d'une donnée est le temps maximal entre deux opérateurs du graphe du data - path (traitement pipeline). Ensuite, nous quantifions le temps restant disponible entre la dernière étape et la contrainte de temps afin d'affiner notre première estimation. La détermination de ce nombre peut être réalisée de manière heuristique parce que son évaluation est réalisée en considérant le pire cas et que dans certains cas une étape supplémentaire peut être incluse. La définition des sous-ensembles $\{O_j,\ldots,O_k\}$ représentant les partitions est loin d'avoir une solution unique. La détermination du nombre d'étapes de reconfigurations et du nombre d'opérations maximal par étape réduit fortement l'espace des solutions de paritionnement, rendant l'évaluation systématique envisageable.

Nous émettons les hypothèses suivantes concernant l'application et la technologie FPGA pour formuler le problème du partitionnement en RD. Nous considérons que les données à traiter sont groupées en blocs de N données. Nous supposons que les mêmes opérateurs sont appliqués sur toutes les données d'un bloc. Nous considérons également que la vitesse de reconfiguration de la technologie FPGA utilisée peut être approximée par une fonction linéaire entre la surface et le temps mis pour la configurer, ainsi la vitesse de configuration est constante.

3. Application pour le traitement temps réel d'images (TI).

Nous illustrons notre méthodologie de partitionnement temporel sur deux data - path d'algorithmes de traitement d'images. Ce domaine convient parfaitement à notre approche, car les données sont naturellement organisées en blocs (les images), de nombreux traitements bas niveau peuvent se modéliser sous forme de graphe flot de données et la contrainte de temps correspond en général à la période d'acquisition images. Nous supposons que les images en niveaux de gris (codés sur 8 bits) sont acquises à raison de 25 par seconde avec une résolution spatiale de 512x512 pixels. Cela nous conduit à traiter une image entière en 40 ms (contrainte de temps).

Le premier traitement est un filtre médian suivi d'un détecteur de Sobel dont le graphe flot de données est représenté sur la figure 1. Ce filtre médian permet d'affecter au pixel central du voisinage 3x3 la valeur médianne des niveaux de gris [5]. Cet opérateur est constitué de comparateurs et multiplexeurs 8 bits. Le calcul du gradient est réalisé par un opérateur de Sobel. Cet opérateur est décomposé en deux filtres monodimensionnels, l'opérateur de Sobel vertical et l'opérateur horizontal. La valeur du gradient du pixel central est le maximum de la valeur absolue du gradient horizontal et vertical.

Le second traitement est un estimateur de mouvement des contours horizontaux (flot optique) utilisé dans un algorithme de détection de mouvement [6].

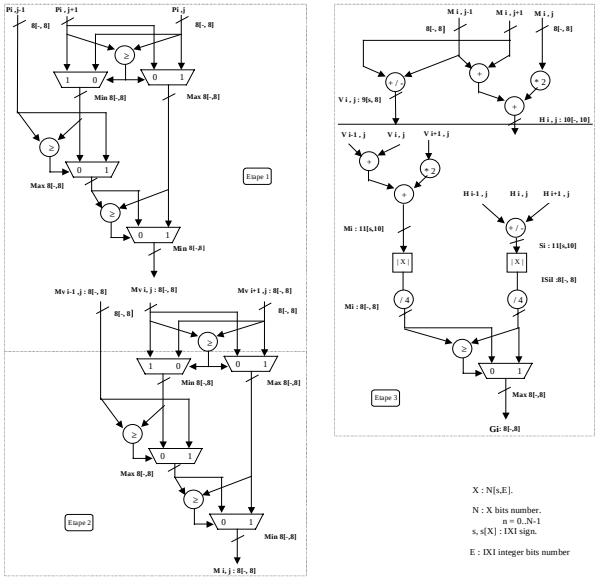

FIG. 1 : Graphe flot de données d'un calcul de gradient.

Il est composé d'un filtre gaussien suivit d'un moyenneur et de dérivateurs spatial et temporel. La figure 2 présente sa représentation sous forme graphe flot de données.

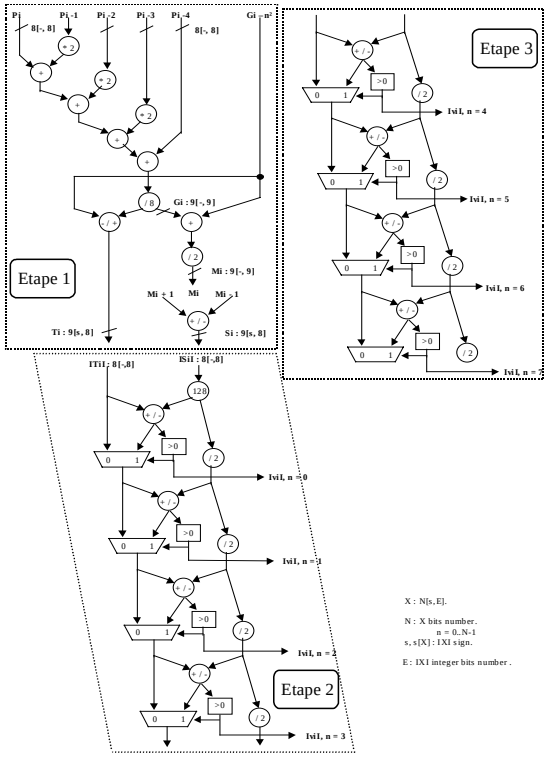

FIG. 2 : Graphe flot de données d'un estimateur du flot optique normal.

Pour vérifier la validité de notre méthode d'adéquation algorithme–architecture, nous appliquons notre méthode sur ces deux exemples simples d'algorithmes TI que l'on souhaite implanter dans un circuit FPGA AT40K d'Atmel. Le travail a été réalisé sur la carte « Ardoise » qui contient un FPGA AT40K20 [7, 8]. Cette famille de circuit supporte la reconfiguration partielle et dispose d'une reconfiguration

totale du circuit inférieure à une milliseconde. Dans le domaine du traitement d'images, ce temps de reconfiguration est assez faible par rapport à la durée de traitement autorisée pour un bloc de données. En pratique la vitesse de configuration est proportionnelle entre le temps de configuration et le nombre de cellules logiques (Cells) dans le FPGA. Chaque temps de configuration dépend de la quantité de cellules logiques utilisée dans chaque étape. Dans notre cas, le AT40K20 d'une capacité de 819 Cells dispose d'un temps de reconfiguration total inférieur de 0.6 ms à 33 MHz avec 8 bits de données de configuration [8]. Nous obtenons une vitesse de configuration de 1365 Cell / ms.

Nous présentons l'évaluation et les résultats expérimentaux des implantations réalisées. L'étude de la structure des cellules logiques de la technologie utilisée permet une évaluation des ressources logiques nécessaires de l'application à partir de son GFD. Par exemple, un additionneur ou soustracteur de n bits synchronisé ou non nécessite n cells. Le même nombre de Cells est nécessaire pour un multiplexeur (non synchronisé) ou un registre. Cela permet l'évaluation de ressources nécessaires pour chaque étape de l'application à partir de son GFD. La détermination des temps d'exécution des opérateurs est obtenue à partir de paramètres caractéristiques de la technologie et de la structure des opérateurs [8]. Dans notre premier exemple, nous savons que les comparateurs sont les opérateurs les plus lents de notre chemin de données (figure 1). Dans notre application, l'opérateur le plus lent est un comparateur synchronisé de taille 8 bits. L'équation suivante donne le temps d'exécution d'un comparateur:

$$to_{max} = (2 \cdot D - 1) \cdot T_C + 2 \cdot T_R + T_{Setup} \qquad (2)$$

Où D est la taille maximale de données à traiter, T_C est temps de propagation des fonctions logiques dans une cellule (temps de traversée d'une cellule logique), T_R est le temps de propagation du routage entre les cellules logiques (délai de routage intra-opérateur) et T_{Setup} est le temps de pré positionnement des registres contenus dans les cellules logiques. Pour un circuit de la famille AT40k en version –2 alimenté en 5V nous obtenons les valeurs suivantes pour ces différents paramètres: T_C = 1.7 ns; T_R = 0.17 ns et T_{Setup} = 1.5 ns [8].

De même, dans le second exemple, à partir du graphe de la figure 2 nous pouvons aisément déterminer que pour la même taille (nombre de bits) les opérateurs les plus lents de notre chemin de données sont les additionneurs avec mémorisation du résultat (fonctionnement en pipeline). L'opérateur le plus lent de cet algorithme est un additionneur synchronisé 15 bits. L'équation suivante donne le temps d'exécution maximal pour un additionneur synchronisé (opérateur cascadé synchronisé):

$$to_{max} = D \cdot (Tc + Tr) + Tsetup \qquad (3)$$

L'expérience acquise en implantation de graphe flot de données des applications de type chemin de données sur ce type de circuits nous permet d'estimer le coefficient de délai de routage inter opérateurs à K=1,5. Ceci est d'autant plus justifié que nous souhaitons effectuer des implantations homogènes en terme de ressources pour chaque étape de traitement (taux d'occupation quasi identique à chaque étape). Nous estimons les données du tableau 1 et 2 pour une implantation RD optimisée de ces deux algorithmes à partir des équations (1), (2) et (3) pour ces deux exemples. Ces résultats sont obtenus avec des images de taille 512 par 512 pixels.

TAB. 1 : Estimation de l'implantation RD d'un détecteur de contours.

Nombre total de Cells	$\max(t_{oi})$ (ns)	Nombre d'étapes	Nombre de Cells / étape	Temps reconfiguration / étape (µs)
467	41	3	156	114

TAB. 2 : Estimation de l'implantation RD d'un estimateur du flot optique.

Nombre total de Cells	$\max(t_{oi})$ (ns)	Nombre d'étapes	Nombre de Cells / étape	Temps reconfiguration / étape (µs)
863	44. 3	3. 27	264	200

Le partitionnement final est choisi de façon à obtenir approximativement le même nombre de cellules logiques dans chaque étape (représenté en pointille en figure 1 et 2). Les tableaux 3 et 4 montrent les résultats expérimentaux obtenus avec notre implantation en RD. Les performances comme le temps de traitement et la quantité de ressources utilisée sont décrites.

TAB. 3 : Résultats de l'implantation du détecteur de contours.

Etape	Nombre de Cells	$\max(t_{oi})$ (ns)	Temps reconfiguration (µs)	Temps de traitement (ms)
1	152	40. 1	111	10. 5
2	156	40 .3	114	10. 6
3	159	36. 7	116	9. 6

Les performances obtenues en terme d'optimisation de surface et de respect des contraintes de temps dépendent du partitionnement de l'algorithme. Nous notons que l'exécution dynamique en trois étapes peut être réalisée en temps- réel (40 ms en traitement d'images). Ceci est en concordance avec nos estimations (3 et 3.27 étapes). Il existe plusieurs manières de partitionner l'algorithme en E étapes. Evidemment, la meilleure solution est de trouver une partition qui conduit au même nombre de cells utilisées dans chaque étape et qui prend en compte les performances de la mémoire utilisée pour le stockage des données intermédiaires entre les étapes.

TAB. 4 : Résultats de l'implantation de l'estimateur du flot optique.

Etape	Nombre de Cells	$\max(t_{oi})$ (ns)	Temps reconfiguration (µs)	Temps de traitement (ms)
1	209	27. 1	160	7. 1
2	354	38. 7	260	10. 15
3	336	37. 8	250	9. 91

A partir du nombre maximal de cells par étape, nous déterminons respectivement les valeurs 3 et 2.43 de gain en terme de densité fonctionnelle. Une partition, qui ne respecte pas les fonctions haut–niveau de l'algorithme, permet une implantation plus homogène en terme de ressources utilisées par chaque étape. Dans le dernier exemple, une meilleure décomposition du data–path en trois étapes sera obtenue avec un nombre moyen de Cells/étape de 288. La valeur de E (nombre d'étapes) est calculée en considérant que chaque étape nécessite une reconfiguration globale du data-path et est exécutée avec la plus faible fréquence de traitement. En pratique, nous avons des temps de reconfiguration et d'exécution inférieurs ou égaux aux temps évalués. C'est pourquoi, quatre partitions sont possibles au lieu d'une valeur théorique de 3.27. Si nous supposons une étape supplémentaire avec un $toi_{\max}$ égal à 44.3 ns (tableau 2), nous trouvons un temps d'exécution total de 11.6 ms qui nous conduit à un temps de reconfiguration suffisant de 1.2 ms pour assurer la contrainte temps-réel. Ceci est compatible avec notre application et permettra d'améliorer la densité fonctionnelle de l'implantation. Une implantation réelle avec un partitionnement homogène en quatre étapes serait donc faisable. Ceci est possible parce que le nombre d'étapes est calculé en considérant le temps d'exécution de l'opérateur le plus lent de l'algorithme à chaque étape ce qui est le pire cas en terme de consommation de ressources temporelles. Avec ce cas, nous montrons que cette approche peut être réalisée de manière heuristique dans le but d'améliorer la précision du nombre d'étapes possibles.

4. Conclusion et perspectives.

Nous avons proposé un nouveau formalisme en vue d'obtenir des implantations respectant des contraintes de temps réel tout en minimisant le matériel. Nous proposons une méthode de partitionnement temporel d'un graphe flot de données qui permet de minimiser la taille de la matrice FPGA en utilisant la reconfiguration dynamique. Pour cela, nous avons proposé une méthode pour évaluer le nombre d'étapes exécution – reconfiguration (E). A partir de l'analyse du GFD, nous déduisons les ressources nécessaires et la vitesse des différents opérateurs. Cela conduit à déterminer le temps de traitement total à partir duquel nous déduisons le partitionnement optimisé du GFD pour une implantation en RD. Les résultats expérimentaux de nos implantations sont en concordance avec nos estimations. Les performances obtenues sont compatibles avec les besoins du traitement temps-réel. Cette méthode peut être abordée de manière heuristique. Plusieurs partitionnements, avec une même quantité de ressources sont possibles. Ainsi un programme heuristique peut nous aider à trouver un partitionnement optimisé suivant des critères établis tels que la taille des données intermédiaires. Notre objectif est de trouver une méthodologie d'implantation en RD générale qui est indépendante du choix du FPGA mais qui utilise ses caractéristiques technologiques. Le but final est de réaliser un procédé automatisé de cette méthode. D'autres algorithmes de data–path plus complexes seront implantés en exploitant cette approche dans le but de valider entièrement ce concept.

Références

[1] M. J. Wirthlin, B.L. Hutchings, "Improving functional density through run-time constant propagation". *FCCM '* 1997.

[2] H. Guermoud, S. Weber, Y. Berviller, E. Tisserand, "Reconfiguration Dynamiquement des FPGA-SRAM pour l'optimisation d'architectures matérielles dédiées aux traitements d'images temps réel" *4ème Journées Adéquation - Algorithmes Architecture en traitement du signal et image*. 28-29-30 janvier 1998.

[3] M. Kaul, R. Vemuri, "Optimal temporal partitioning and synthesis for reconfigurable architectures" Int. Symposium on Field-Programmable Custom Computing Machines, pages 312-313, April 1998.

[4] W. Luk, N. Shirazi, P.Y.K. Cheung, "Modelling and Optimising Run-Time Reconfiguration Systems", *in Proc. IEEE Symposium on FPGAs for Custom Computing Machines*, K.L. Pocek and J. Arnold (editors), *IEEE Computer Society Press*, 1996, pp. 167-176.

[5] N. Demassieux. Architecture VLSI pour le traitement d'images : une contribution à l'étude du traitement matériel de l'information. Thèse de doctorat de l'école nationale supérieure des télécommunications (ENST), 1991.

[6] C. Tanougast, Y. Berviller, S. Weber " Optimization of Motion Estimator for Run-Time-Reconfiguration Implementation ", Lecture Notes in Computer Science, Vol. 1800, p. 959-965, Parallel and Distributed Processing, J. Rolim Editor, Springer Verlag, 2000.

[7] D. Demigny, M. Paindavoine, S. Weber, «Architecture Reconfigurable Dynamiquement pour le Traitement Temps Réel des Images ». *Revue technique et Sciences de l'information, Numéro Spécial programmation des Architectures Reconfigurables.* 1998.

[8] Atmel. AT40K FPGA. Data Sheet.